

Chapter 3: An SQL server tutorial

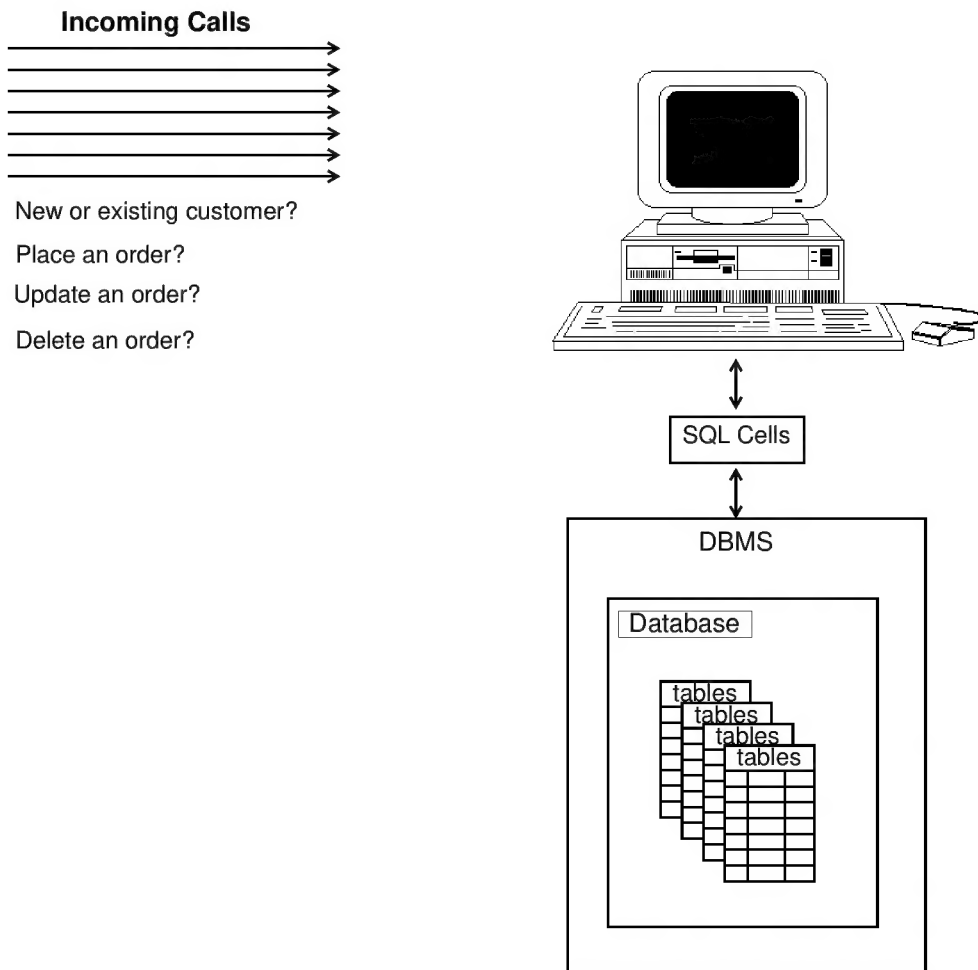
This chapter briefly describes an application that accesses an Ingres database and shows the use of IVR 2.0/I SQL cells.

The following paragraph describes a typical scenario of a distributor who sells clothing by catalog. The application is being implemented to streamline order processing.

Michelle Jordon is an application developer at Kendall and Ives, Inc., a national clothing distributor that uses catalog sales to distribute its products. Michelle is in charge of developing an IVR application to provide high-quality service to customers calling in orders or requesting the status of their orders.

The application will access a database called “catalog,” which contains customer, product, and order information. Customers can call an 800 number to access the company’s Customer Service Division to place an order, check order status, update an order, or delete an order.

Figure 3-1
Sample IVR catalog ordering application



When Michelle defines the default cell parameter, she identifies the SQL DBMS type as "INGRES" and the database name as "catalog".

Figure 3-2
DBMS type and database name parameters

The screenshot shows a dialog box titled "DFLT Parameters". It contains several configuration fields:

- A numeric field with the value "200" and a label "Minimum Voice Duration (ms)".
- A text field for "SQL DBMS Type" with the value "Ingres".
- A text field for "SQL Database Name" with the value "Catalog".
- A numeric field with the value "1" and a label "SQL Maximum Server Count".
- A text field for "User Name" with the value "dbuser".
- A text field for "Password" with masked characters "*****".
- A text field for "Host Name" with the value "<NONE>".

At the bottom of the dialog box are three buttons: "Apply", "Cancel", and "Help".

One of the first tasks the application will do will be to query the database to see if the caller is an existing customer. To do this, Michelle creates a QSEL cell to search the customer table for all entries where the value of the "phone" field is equal to the telephone number that the caller enters in response to an earlier GDAT cell.

If there are no entries (that is, no rows are selected), the caller is identified as a new customer. If an error occurs while the cell is being processed, the application will play the message and transfer to a live operator. Figure 3-3 shows the QSEL cell created and its parameters, the next cell specified, and the table that is being queried.

Figure 3-3
QSEL Parameters

customer

customer	phone	credit card #	cc type
Jerry Calert	(508)2645527	543 7589 3445	VISA
Paul Abram	(617)2705903	267 8954 2334	MC
Mary Jones	(508)9675096	432 9087 5667	AMEX
Janet Johns	(603)5872584	498 9765 4889	VISA

QSEL cell accesses this table

Cell #4 **QSEL SQL Select**

Retrieve Row:

Comments:

Call Audit Enabled? ☐ Yes ☒ No

Call Audit Information:

SQL Table Name:

Column and Buffer Table

Column	Buffer
price	DATA EXCHANGE #1
	DATA EXCHANGE #1
	DATA EXCHANGE #1
	DATA EXCHANGE #1
	DATA EXCHANGE #1

▼ More ▲ Less

Where Clause

Logical	's	Column	Type	Operator	Value	's
	((	order	Numeric	>	1	)
And	(	status	String	=	pending	)
			String		DATA EXCHANGE #1	

Apply Cancel Help

If the caller is an existing customer, the application checks to see if there are any outstanding orders. For this transaction, Michelle creates a QCNT cell to count the number of rows in the “order_header” table where the “customer_id” field is equal to the customer’s telephone number and the status field is not equal to “d” (for delivered). Figure 3-4 shows the parameters for this cell, the corresponding embedded SQL statement processed, and the table that is queried.

Figure 3-4
QCNT Cell for executing a SELECT COUNT statement

Corresponding SQL statement

```
SELECT COUNT (*)
FROM order_header
WHERE ((customer_id = :telephone_number)
AND (status != "d"));
```

QCNT cell accesses this table

order_header		
customer	phone #	status
5085527	(508)2645527	d
6175903	(617)5903267	p
5085096	(508)9675096	d
6032584	(603)5872584	d



QCNT Parameters

Cell #1 **QCNT** SQL Select Count

Count Orders

Comments Count pending orders

Call Audit Enabled? ☒ Yes ☐ No

Call Audit Information DIGITS

SQL Table Name "order_header"

Logical	{'s	Column	Type	Operator	Value	}'s
	{	customer_id	Numeric	=	telephone_number	}
And	{	status	String	!=	"d"	}
			String		DATA EXCHANGE #1	
			String		DATA EXCHANGE #1	

Apply Cancel Help

Updating an existing order

To allow a customer to update an existing order, Michelle creates a QUPD cell, as shown in Figure 3-5. This call will execute this embedded SQL UPDATE statement:

Figure 3-5
Identifying the table to be updated with the QUPD cell

Corresponding SQL statement

```
UPDATE order_header
SET product_num=BUFFER1, color=BUFFER2, size=BUFFER3
WHERE order_num=digits
```

QUPD cell accesses this table

order_header				
order number	customer	product #	color	size
200567	5085527	495	blue	XL
200897	6175903	267	red	SM
200598	5085096	432	white	L
200607	6032584	498	black	M

Cell #5 **QUPD SQL Update**

Update Order

Comments: Customer can update

Call Audit Enabled? ☒ Yes ☐ No

Call Audit Information: DIGITS

SQL Table Name: order_header

Column	Type	Value
product_num	Numeric	BUFFER1
color	String	BUFFER2
size	String	BUFFER3
	String	DATA EXCHANGE #1

▼ More ▲ Less

Logical	Column	Type	Operator	Value	}'s
	order_num	Numeric	=	DIGITS	
		String		DATA EXCHANGE #1	
		String		DATA EXCHANGE #1	
		String		DATA EXCHANGE #1	

Apply Cancel Help

Deleting an order

The application that Michelle is designing with IVR 2.0/I must also permit a caller to delete an order. With the QDEL cell, Michelle can execute a DELETE statement to remove the appropriate row from the order_header table as shown in Figure 3-6. The cell will execute the embedded SQL statement.

Figure 3-6
Identifying the table to be accessed by the QDEL cell

Corresponding SQL statement

```
DELETE FROM order_header
WHERE order_num=order_number
```

QDEL cell accesses this table

order_header				
order number	customer	product #	color	size
200567	5085527	495	blue	XL
200897	6175903	267	red	SM
200598	5085096	432	white	L
200607	8032584	498	black	M

QDEL Parameters

Cell #2 **QDEL SQL Delete**

Delete Order

Comments: Caller can delete order

Call Audit Enabled? ☒ Yes ☐ No

Call Audit Information: DIGITS

SQL Table Name: order_header

Logical	('s	Column	Type	Operator	Value	) 's
		order_num	Numeric	=	order_num	
			String		DATA EXCHANGE #1	
			String		DATA EXCHANGE #1	
			String		DATA EXCHANGE #1	

More Less

Apply Cancel Help

Inserting information into the database

Throughout the application, Michelle includes the QINS cell to insert data into different tables in the CATALOG database. For example, when the caller decides to place a new order, the QINS cell shown in Figure 3-7 inserts a new row in the order_header table. This call loads different values in the order_num, customer_id, status, and op_initials columns, corresponding to the embedded SQL statement.

Figure 3-7
Identifying the table to be accessed by the QINS cell

Corresponding SQL statement

```
INSERT INTO order_header (order_num, customer_id, status, op_initials)
VALUES (:cust, :order_num, :status, :operator);
```

QINS cell accesses this table

order_header			
order_num	customer_id	status	op_initials
200567	5085527	d	jr
200697	6175903	p	jb
200598	5085096	d	pg
200607	6032584	d	jr

Cell #3 **QINS** SQL Insert

Insert Order

Comments: Caller places new order

Call Audit Enabled? ☒ Yes ☐ No

Call Audit Information: DIGITS

SQL Table Name: cust

Column	Type	Value
customer_id	String	CUST
order_num	Numeric	ORDER_NUM
status	String	STATUS
op_initials	String	OPERATOR

More Less

Apply Cancel Help

Later, Michelle uses additional SQL cells to insert order detail, update the order total, and update the number of orders associated with a particular telephone order. She can add new fields as needed in a piecemeal fashion without disrupting the existing users as needs change. If you are familiar with standard and embedded SQL statements, you can quickly design IVR 2.0/I applications to access your database.